C++ Grammar Summary

From the Annex A of ISO/IEC 14882:1998 (The C++ Standard). As it says there, this is not an exact statement of the language.

Go straight to: *translation-unit*.

A.1 Keywords

typedef-name:
    identifier

namespace-name:
    original-namespace-name
    namespace-alias

namespace-alias:
    identifier

class-name:
    identifier
    template-id

enum-name:
    identifier

template-name:
    identifier

A.2 Lexical conventions

hex-quad:
    hexadecimal-digit *hexadecimal-digit* *hexadecimal-digit* *hexadecimal-digit*

universal-character-name:
    \u *hex-quad*
    \U *hex-quad* *hex-quad*

preprocessing-token:
    header-name
    identifier
    pp-number
    character-literal
    string-literal
    preprocessing-op-or-punc
    each non-white-space character that cannot be one of the above

token:
    identifier
    keyword
    literal
    operator
    punctuator

header-name:

< *h-char-sequence* >
" *q-char-sequence* "

h-char-sequence:
h-char
h-char-sequence *h-char*

h-char:
any member of the source character set except new-line and >

q-char-sequence:
q-char
q-char-sequence *q-char*

q-char:
any member of the source character set except new-line and "

pp-number:
digit
. *digit*
pp-number *digit*
pp-number *nondigit*
pp-number e *sign*
pp-number E *sign*
pp-number .

identifier:
nondigit
identifier *nondigit*
identifier *digit*

nondigit: one of
universal-character-name
_ a b c d e f g h i j k l m
n o p q r s t u v w x y z
A B C D E F G H I J K L M
N O P Q R S T U V W X Y Z

digit: one of
0 1 2 3 4 5 6 7 8 9

preprocessing-op-or-punc: one of
{ } [ ] # ## ( )
<: :> <% %> %: %:%: ; : ...
new delete ? :: . .*
+ - * / % ^ & | _
! = < > += -= *= /= %=
^= &= |= << >> <<= >>= == !=
<= >= && || ++ -- , ->* ->
and and_eq bitand bitor compl not not_eq
or or_eq xor xor_eq

literal:
integer-literal
character-literal
floating-literal
string-literal

boolean-literal

integer-literal:
decimal-literal *integer-suffix*$_{opt}$
octal-literal *integer-suffix*$_{opt}$
hexadecimal-literal *integer-suffix*$_{opt}$

decimal-literal:
nonzero-digit
decimal-literal *digit*

octal-literal:
`0`
octal-literal *octal-digit*

hexadecimal-literal:
`0x` *hexadecimal-digit*
`0X` *hexadecimal-digit*
hexadecimal-literal *hexadecimal-digit*

nonzero-digit: one of
`1 2 3 4 5 6 7 8 9`

octal-digit: one of
`0 1 2 3 4 5 6 7`

hexadecimal-digit: one of
`0 1 2 3 4 5 6 7 8 9`
`a b c d e f`
`A B C D E F`

integer-suffix:
unsigned-suffix *long-suffix*$_{opt}$
long-suffix *unsigned-suffix*$_{opt}$

unsigned-suffix: one of
`u U`

long-suffix: one of
`l L`

character-literal:
`'` *c-char-sequence* `'`
`L'` *c-char-sequence* `'`

c-char-sequence:
c-char
c-char-sequence *c-char*

c-char:
any member of the source character set except the single-quote ', backslash \, or new-line character
escape-sequence
universal-character-name

escape-sequence:
simple-escape-sequence
octal-escape-sequence

hexadecimal-escape-sequence

simple-escape-sequence: one of
`\' \" \? \\ \a \b \f \n \r \t \v`

octal-escape-sequence:
\ *octal-digit*
\ *octal-digit* *octal-digit*
\ *octal-digit* *octal-digit* *octal-digit*

hexadecimal-escape-sequence:
`\x` *hexadecimal-digit*
hexadecimal-escape-sequence *hexadecimal-digit*

floating-literal:
fractional-constant *exponent-part*$_{opt}$ *floating-suffix*$_{opt}$
digit-sequence *exponent-part*$_{opt}$ *floating-suffix*$_{opt}$

fractional-constant:
digit-sequence$_{opt}$. *digit-sequence*
digit-sequence .

exponent-part:
`e` *sign*$_{opt}$ *digit-sequence*
`E` *sign*$_{opt}$ *digit-sequence*

sign: one of
`+ -`

digit-sequence:
digit
digit-sequence *digit*

floating-suffix: one of
`f l F L`

string-literal:
" *s-char-sequence* "
`L`" *s-char-sequence* "

s-char-sequence:
s-char
s-char-sequence *s-char*

s-char:
any member of the source character set except the double-quote ", backslash \, or new-line character
escape-sequence
universal-character-name

boolean-literal:
`false`
`true`

A.3 Basic concepts

translation-unit:

declaration-seq$_{opt}$

A.4 Expressions

primary-expression:
- *literal*
- `this`
- `(` *expression* `)`
- *id-expression*

id-expression:
- *unqualified-id*
- *qualified-id*

unqualified-id:
- *identifier*
- *operator-function-id*
- *conversion-function-id*
- `~` *class-name*
- *template-id*

qualified-id:
- `::`$_{opt}$ *nested-name-specifier* `template`$_{opt}$ *unqualified-id*
- `::` *identifier*
- `::` *operator-function-id*
- `::` *template-id*

nested-name-specifier:
- *class-or-namespace-name* `::` *nested-name-specifier*$_{opt}$
- *class-or-namespace-name* `::` `template` *nested-name-specifier*

class-or-namespace-name:
- *class-name*
- *namespace-name*

postfix-expression:
- *primary-expression*
- *postfix-expression* `[` *expression* `]`
- *postfix-expression* `(` *expression-list*$_{opt}$ `)`
- *simple-type-specifier* `(` *expression-list*$_{opt}$ `)`
- `typename` `::`$_{opt}$ *nested-name-specifier* *identifier* `(` *expression-list*$_{opt}$ `)`
- `typename` `::`$_{opt}$ *nested-name-specifier* `template`$_{opt}$ *template-id* `(` *expression-list*$_{opt}$ `)`
- *postfix-expression* `.` `template`$_{opt}$ *id-expression*
- *postfix-expression* `->` `template`$_{opt}$ *id-expression*
- *postfix-expression* `.` *pseudo-destructor-name*
- *postfix-expression* `->` *pseudo-destructor-name*
- *postfix-expression* `++`
- *postfix-expression* `--`
- `dynamic_cast` `<` *type-id* `>` `(` *expression* `)`
- `static_cast` `<` *type-id* `>` `(` *expression* `)`
- `reinterpret_cast` `<` *type-id* `>` `(` *expression* `)`
- `const_cast` `<` *type-id* `>` `(` *expression* `)`
- *type-id* `(` *expression* `)`
- *type-id* `(` *type-id* `)`

expression-list:

assignment-expression
expression-list , *assignment-expression*

pseudo-destructor-name:
::$_{opt}$ *nested-name-specifier*$_{opt}$ *type-name* :: ~ *type-name*
::$_{opt}$ *nested-name-specifier* template *template-id* :: ~ *type-name*
::$_{opt}$ *nested-name-specifier*$_{opt}$ ~ *type-name*

unary-expression:
postfix-expression
++ *cast-expression*
-- *cast-expression*
unary-operator *cast-expression*
sizeof *unary-expression*
sizeof (*type-id*)
new-expression
delete-expression

unary-operator: one of
* & + - ! ~

new-expression:
::$_{opt}$ new *new-placement*$_{opt}$ *new-type-id* *new-initializer*$_{opt}$
::$_{opt}$ new *new-placement*$_{opt}$ (*type-id*) *new-initializer*$_{opt}$

new-placement:
(*expression-list*)

new-type-id:
type-specifier-seq *new-declarator*$_{opt}$

new-declarator:
ptr-operator *new-declarator*$_{opt}$
direct-new-declarator

direct-new-declarator:
[*expression*]
direct-new-declarator [*constant-expression*]

new-initializer:
(*expression-list*$_{opt}$)

delete-expression:
::$_{opt}$ delete *cast-expression*
::$_{opt}$ delete [ ] *cast-expression*

cast-expression:
unary-expression
(*type-id*) *cast-expression*

pm-expression:
cast-expression
pm-expression .* *cast-expression*
pm-expression ->* *cast-expression*

multiplicative-expression:
pm-expression

multiplicative-expression * *pm-expression*
multiplicative-expression / *pm-expression*
multiplicative-expression % *pm-expression*

additive-expression:
multiplicative-expression
additive-expression + *multiplicative-expression*
additive-expression - *multiplicative-expression*

shift-expression:
additive-expression
shift-expression << *additive-expression*
shift-expression >> *additive-expression*

relational-expression:
shift-expression
relational-expression < *shift-expression*
relational-expression > *shift-expression*
relational-expression <= *shift-expression*
relational-expression >= *shift-expression*

equality-expression:
relational-expression
equality-expression == *relational-expression*
equality-expression != *relational-expression*

and-expression:
equality-expression
and-expression & *equality-expression*

exclusive-or-expression:
and-expression
exclusive-or-expression ^ *and-expression*

inclusive-or-expression:
exclusive-or-expression
inclusive-or-expression | *exclusive-or-expression*

logical-and-expression:
inclusive-or-expression
logical-and-expression && *inclusive-or-expression*

logical-or-expression:
logical-and-expression
logical-and-expression || *logical-or-expression*

conditional-expression:
logical-or-expression
logical-or-expression ? *expression*
: *assignment-expression*

assignment-expression:
conditional-expression
logical-or-expression *assignment-operator* *assignment-expression*
throw-expression

assignment-operator: one of

`= *= /= %= += -= <<= >>= &= ^= |=`

expression:
assignment-expression
expression `,` *assignment-expression*

constant-expression:
conditional-expression

A.5 Statements

statement:
labeled-statement
expression-statement
compound-statement
selection-statement
iteration-statement
jump-statement
declaration-statement
try-block

labeled-statement:
identifier `:` *statement*
`case` *constant-expression* `:` *statement*
`default :` *statement*

expression-statement:
expression$_{opt}$ `;`

compound-statement:
`{` *statement-seq*$_{opt}$ `}`

statement-seq:
statement
statement-seq *statement*

selection-statement:
`if (` *condition* `)` *statement*
`if (` *condition* `)` *statement* `else` *statement*
`switch (` *condition* `)` *statement*

condition:
expression
type-specifier-seq *declarator* `=` *assignment-expression*

iteration-statement:
`while (` *condition* `)` *statement*
`do` *statement* `while (` *expression* `) ;`
`for (` *for-init-statement* *condition*$_{opt}$ `;` *expression*$_{opt}$ `)` *statement*

for-init-statement:
expression-statement
simple-declaration

jump-statement:
`break ;`
`continue ;`

`return` *expression*$_{opt}$ `;`
`goto` *identifier* `;`

declaration-statement:
block-declaration

A.6 Declarations

declaration-seq:
declaration
declaration-seq *declaration*

declaration:
block-declaration
function-definition
template-declaration
explicit-instantiation
explicit-specialization
linkage-specification
namespace-definition

block-declaration:
simple-declaration
asm-definition
namespace-alias-definition
using-declaration
using-directive

simple-declaration:
decl-specifier-seq$_{opt}$ *init-declarator-list*$_{opt}$ `;`

decl-specifier:
storage-class-specifier
type-specifier
function-specifier
`friend`
`typedef`

decl-specifier-seq:
decl-specifier-seq$_{opt}$ *decl-specifier*

storage-class-specifier:
`auto`
`register`
`static`
`extern`
`mutable`

function-specifier:
`inline`
`virtual`
`explicit`

typedef-name:
identifier

type-specifier:

simple-type-specifier
class-specifier
enum-specifier
elaborated-type-specifier
cv-qualifier

simple-type-specifier:
`::`*opt* *nested-name-specifier*_opt *type-name*
`::`*opt* *nested-name-specifier* `template` *template-id*
`char`
`wchar_t`
`bool`
`short`
`int`
`long`
`signed`
`unsigned`
`float`
`double`
`void`

type-name:
class-name
enum-name
typedef-name

elaborated-type-specifier:
class-key `::`*opt* *nested-name-specifier*_opt *identifier*
`enum` `::`*opt* *nested-name-specifier*_opt *identifier*
`typename` `::`*opt* *nested-name-specifier* *identifier*
`typename` `::`*opt* *nested-name-specifier* `template`*opt* *template-id*

enum-name:
identifier

enum-specifier:
`enum` *identifier*_opt `{` *enumerator-list*_opt `}`

enumerator-list:
enumerator-definition
enumerator-list `,` *enumerator-definition*

enumerator-definition:
enumerator
enumerator `=` *constant-expression*

enumerator:
identifier

namespace-name:
original-namespace-name
namespace-alias

original-namespace-name
identifier

namespace-definition:

named-namespace-definition
unnamed-namespace-definition

named-namespace-definition:
original-namespace-definition
extension-namespace-definition

original-namespace-definition:
`namespace` *identifier* `{` *namespace-body* `}`

extension-namespace-definition:
`namespace` *original-namespace-name* `{` *namespace-body* `}`

unnamed-namespace-definition:
`namespace` `{` *namespace-body* `}`

namespace-body:
declaration-seq$_{opt}$

namespace-alias:
identifier

namespace-alias-definition:
`namespace` *identifier* `=` *qualified-namespace-specifier* `;`

qualified-namespace-specifier:
`::`$_{opt}$ *nested-name-specifier*$_{opt}$ *namespace-name*

using-declaration:
`using` `typename`$_{opt}$ `::`$_{opt}$ *nested-name-specifier* *unqualified-id* `;`
`using` `::` *unqualified-id* `;`

using-directive:
`using` `namespace` `::`$_{opt}$ *nested-name-specifier*$_{opt}$ *namespace-name* `;`

asm-definition:
`asm` `{` *string-literal* `}`

linkage-specification:
`extern` *string-literal* `{` *declaration-seq*$_{opt}$ `}`
`extern` *string-literal* *declaration*

A.7 Declarators

init-declarator-list:
init-declarator
init-declarator-list *init-declarator*

init-declarator:
declarator *initializer*$_{opt}$

declarator:
direct-declarator
ptr-operator *declarator*

direct-declarator:
declarator-id

direct-declarator (*parameter-declaration-clause*) *cv-qualifier-seq*$_{opt}$ *exception-specification*$_{opt}$
direct-declarator [*constant-expression*$_{opt}$]
(*declarator*)

ptr-operator:
* *cv-qualifier-seq*$_{opt}$
&
::$_{opt}$ *nested-name-specifier* * *cv-qualifier-seq*$_{opt}$

cv-qualifier-seq:
cv-qualifier *cv-qualifier-seq*$_{opt}$

cv-qualifier:
`const`
`volatile`

declarator-id:
::$_{opt}$ *nested-name-specifier*$_{opt}$ *type-name*

type-id:
type-specifier-seq *abstract-declarator*$_{opt}$

type-specifier-seq:
type-specifier *type-specifier-seq*$_{opt}$

abstract-declarator:
ptr-operator *abstract-declarator*$_{opt}$
direct-abstract-declarator

direct-abstract-declarator:
direct-abstract-declarator$_{opt}$ (*parameter-declaration-clause*) *cv-qualifier-seq*$_{opt}$ *exception-specification*$_{opt}$
direct-abstract-declarator$_{opt}$ [*constant-expression*$_{opt}$]
(*abstract-declarator*)

parameter-declaration-clause:
parameter-declaration-list$_{opt}$ ...$_{opt}$
parameter-declaration-list , ...

parameter-declaration-list:
parameter-declaration
parameter-declaration-list , *parameter-declaration*

parameter-declaration:
decl-specifier-seq *declarator*
decl-specifier-seq *declarator* = *assignment-expression*
decl-specifier-seq *abstract-declarator*$_{opt}$
decl-specifier-seq *abstract-declarator*$_{opt}$ = *assignment-expression*

function-definition:
decl-specifier-seq$_{opt}$ *declarator* *ctor-initializer*$_{opt}$ *function-body*
decl-specifier-seq$_{opt}$ *declarator* *function-try-block*

function-body:
compound-statement

initializer:
= *initializer-clause* (*expression-list*)

initializer-clause:
assignment-expression
{ *initializer-list* ,*opt* }
{ }

initializer-list:
initializer-clause
initializer-list , *initializer-clause*

A.8 Classes

class-name:
identifier
template-id

class-specifier:
class-head { *member-specification*$_{opt}$ }

class-head:
class-key *identifier*$_{opt}$ *base-clause*$_{opt}$
class-key *nested-name-specifier*$_{opt}$ *identifier* *base-clause*$_{opt}$
class-key *nested-name-specifier*$_{opt}$ *template-id* *base-clause*$_{opt}$

class-key:
class
struct
union

member-specification:
member-declaration *member-specification*$_{opt}$
access-specifier : *member-specification*$_{opt}$

member-declaration:
decl-specifier-seq$_{opt}$ *member-declarator-list*$_{opt}$;
function-definition ;*opt*
::*opt* *nested-name-specifier* template$_{opt}$ *unqualified-id* ;
using-declaration
template-declaration

member-declarator-list:
member-declarator
member-declarator-list , *member-declarator*

member-declarator:
declarator *pure-specifier*$_{opt}$
declarator *constant-initializer*$_{opt}$
identifier$_{opt}$: *constant-expression*

pure-specifier:
= 0

constant-initializer:
= *constant-expression*

A.9 Derieved classes

base-clause:
: `:` *base-specifier-list*

base-specifier-list:
base-specifier
base-specifier-list `,` *base-specifier*

base-specifier:
`::`$_{opt}$ *nested-name-specifier*$_{opt}$ *class-name*
`virtual` *access-specifier*$_{opt}$ `::`$_{opt}$ *nested-name-specifier*$_{opt}$ *class-name*
access-specifier$_{opt}$ `virtual`$_{opt}$ `::`$_{opt}$ *nested-name-specifier*$_{opt}$ *class-name*

access-specifier:
`private`
`protected`
`public`

A.10 Special member functions

conversion-function-id:
`operator` *conversion-type-id*

conversion-type-id:
type-specifier-seq *conversion-declarator*$_{opt}$

conversion-declarator:
ptr-operator *conversion-declarator*$_{opt}$

ctor-initializer:
`:` *mem-initializer-list*

mem-initializer-list:
mem-initializer
mem-initializer-list `,` *mem-initializer*

mem-initializer:
mem-initializer-id `{` *expression-list*$_{opt}$ `}`

mem-initializer-id:
`::`$_{opt}$ *nested-name-specifier*$_{opt}$ *class-name*
identifier

A.11 Overloading

operator-function-id:
`operator` *operator*

operator: one of
```
new delete new[] delete[]
+ - * / % ^ & | ~
! = < > += -= *= /= %=
^= &= |= << >> <<= >>= == !=
<= >= && || ++ -- , ->* ->
() []
```

A.12 Templates

template-declaration:
`export`$_{opt}$ `template` `<` *template-parameter-list* `>` *declaration*

template-parameter-list:
template-parameter
template-parameter-list `,` *template-parameter*

template-parameter:
type-parameter
parameter-declaration

type-parameter:
`class` *identifier*$_{opt}$
`class` *identifier*$_{opt}$ `=` *type-id*
`typename` *identifier*$_{opt}$
`typename` *identifier*$_{opt}$ `=` *type-id*
`template` `<` *template-parameter-list* `>` `class` *identifier*$_{opt}$
`template` `<` *template-parameter-list* `>` `class` *identifier*$_{opt}$ `=` *id-expression*

template-id:
template-name `<` *template-argument-list*$_{opt}$ `>` `class`

template-name:
identifier

template-argument-list:
template-argument
template-argument-list `,` *template-argument*

template-argument:
assignment-expression
type-id
id-expression

explicit-instantiation:
`template` *declaration*

explicit-specialization:
`template` `<` `>` *declaration*

A.13 Exception handling

try-block:
`try` *compound-statement* *handler-seq*

function-try-block:
`try` *ctor-initializer*$_{opt}$ *function-body-handler-seq*

handler-seq:
handler *handler-seq*$_{opt}$

handler:
`catch` `{` *exception-declaration* `}` *compound-statement*

exception-declaration:

type-specifier-seq *declarator*
type-specifier-seq *abstract-declarator*
type-specifier-seq
`...`

throw-expression:
`throw` *assignment-expression*$_{opt}$

exception-specification:
`throw (` *type-id-list*$_{opt}$ `)`

type-id-list:
type-id
type-id-list `,` *type-id*

A.14 Preprocessing directives

preprocessing-file:
group$_{opt}$

group:
group-part
group *group-part*

group-part:
pp-tokens$_{opt}$ *new-line*
if-action
control-line

if-action:
if-group *elif-groups*$_{opt}$ *else-group*$_{opt}$ *endif-line*

if-group:
`# if` *constant-expression* *new-line* *group*$_{opt}$
`# ifdef` *identifier* *new-line* *group*$_{opt}$
`# ifndef` *identifier* *new-line* *group*$_{opt}$

elif-groups:
elif-group
elif-groups *elif-group*

elif-group:
`# elif` *constant-expression* *new-line* *group*$_{opt}$

else-group:
`# else` *new-line* *group*$_{opt}$

endif-line:
`# endif` *new-line*

control-line:
`# include` *pp-tokens* *new-line*
`# define` *identifier* *replacement-list* *new-line*
`# define` *identifier* *lparen* *identifier-list* `)` *replacement-list* *new-line*
`# undef` *identifier* *new-line*
`# line` *pp-tokens* *new-line*

`#` `error` *pp-tokens*$_{opt}$ *new-line*
`#` `pragma` *pp-tokens*$_{opt}$ *new-line*
`#` *new-line*

lparen:
the left-parenthesis character without preceding white-space

replacement-list:
pp-tokens$_{opt}$

pp-tokens:
preprocessing-token
pp-tokens preprocessing-token

new-line:
the new-line character

$LastChangedDate: 2003-03-23 $